Image & Video Processing

Assignment 4 - Color and Multi-scale Representations

Albert Cerfeda

Contents

29 May 2023
Università della Svizzera italiana
Faculty of Informatics
Switzerland

1 Color Palette Extraction from Images [7 points]

1.1 Linear RGB and sRGB Color Palettes

In this exercise we are tasked in obtaining the `sRGB` color space by linearizing the `RGB` color space and extracting a color palette from an image. The color palette is then extracted from both color spaces.

```
function [img_xyz] = rgb2xyz(img_rgb)
    img_xyz = zeros(size(img_rgb));
    mat = [0.4124564 0.3575761 0.1804375;
           0.2126729 0.7151522 0.0721750;
           0.0193339 0.1191920 0.9503041];
    for i = 1:size(img_rgb, 1)
        for j = 1:size(img_rgb, 2)
            rgb = img_rgb(i, j, :);
            img_xyz(i, j, :) = mat * rgb(:);
        end
    end
end
```

Figure 1: Matlab function converting RGB to the sRGB color space

```
function [img_clustered, img_palette, img_layers] = clusterColors(img, n_clusters)
    % 1. Cluster
    A = reshape(img, [], 3);
    [cluster_idx, cluster_center] = kmeans(A, n_clusters);

    % Compute clustered image
    img_clustered = reshape(cluster_center(cluster_idx, :), size(img, 1), size(img, 2), 3);
    % Matrix of cluster indices for corresponding image pixel
    cluster_idx = reshape(cluster_idx, size(img, 1), size(img, 2), 1);

    % 2. Separate image into palette layers
    color_square = zeros(size(img));
    img_palette = {};
    img_layers = {};
    for i = 1:n_clusters
        % Create a squared image with a uniform color using function repmat
        color = cluster_center(i, :);
        color_square(:,:,1) = repmat(color(1), size(img, 1), size(img, 2));
        color_square(:,:,2) = repmat(color(2), size(img, 1), size(img, 2));
        color_square(:,:,3) = repmat(color(3), size(img, 1), size(img, 2));
        layer = color_square .* (cluster_idx == i);

        img_palette{i} = color_square(1,1,:);
        img_layers{i} = layer;
    end
end
```

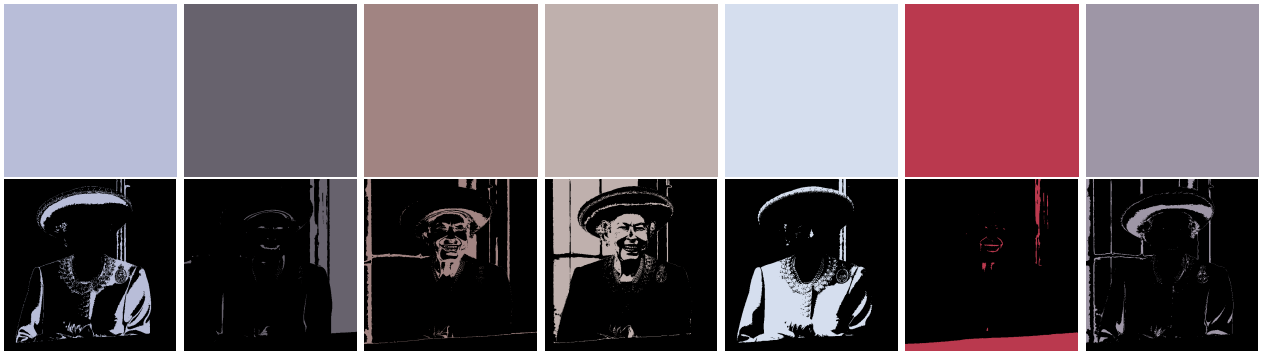
Figure 2: Matlab function performing color quantization

We perform *k-means* clustering[1] on the image colors, by treating each pixel color value as a 3-dimensional point and by letting the algorithm find the $k = 7$ centroids to cluster similar colors together. The centroids are then used as the color palette. The code is shown in Figure 2.
Results are shown in Figure 3a and Figure 3b.

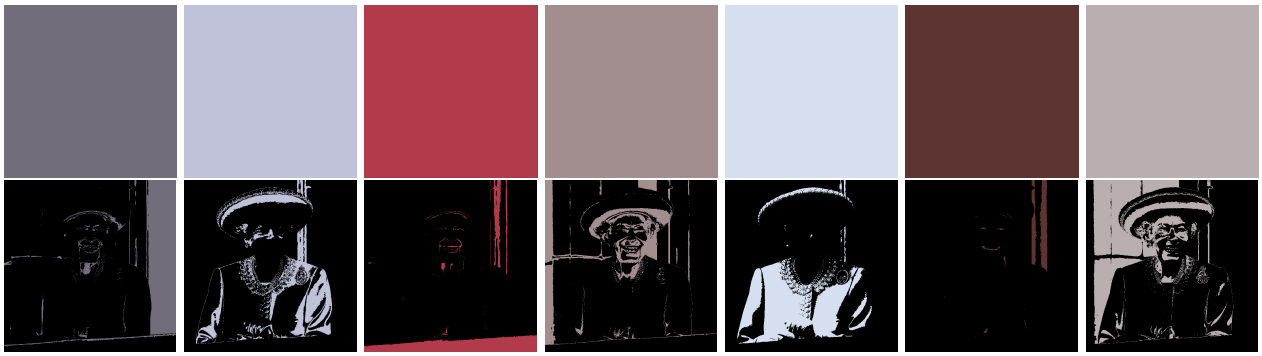
1.2 CIELab Color Palette

RGB is not the best color model for image processing as, since it specifies the quantities of each Red/Blue/Green pigment for obtaining a color, it not provide us with an intuitive way of understanding and performing operations on colors. For example, it is not straightforward to manipulate the saturation or the brightness of a color in RGB

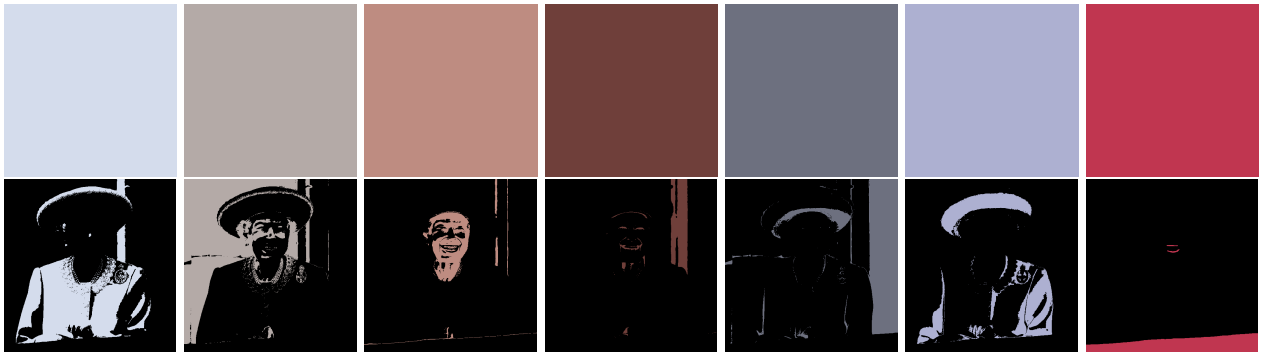
[1] K-means clustering: en.wikipedia.org



(a) RGB Palette and Layers



(b) sRGB Palette and Layers



(c) CIELab Palette and Layers

(d) RGB Clustered image (e) sRGB Clustered image (f) CIELab Clustered image

Figure 3: Color palette extraction using the RGB, sRGB and CIELab color spaces.

space. An alternative to the RGB color space is the HSV[2] color space. It defines each color using 3 channels: Hue, Saturation and Value, allowing intuitive color manipulations. However, the HSV color space is not perceptually uniform, meaning that the distance between two colors in the HSV space does not correspond to the perceived difference between the two colors.

This is where the CIELab[3] comes in handy. The CIELab color space is a perceptually uniform color space, meaning that the distance between two colors in the CIELab space corresponds to the perceived difference between the two color by the human visual system. The CIELab color space is defined by three channels: L*, a* and b*. They

[2]HSV: programmingdesignsystems.com
[3]CIELab: hunterlab.com

represent the lightness of the color (L* = 0 yields black and L* = 100 indicates white), its position between magenta and green (a*, where negative values indicate green and positive values indicate red) and its position between yellow and blue (b*, where negative values indicate blue and positive values indicate yellow).

By clustering colors in the CIELab color space, we will obtain clusters whose points are close to each other in the CIELab space, meaning that they are close to each other in the perceptual space (i.e our eyes perceive these colors as similar), resulting in a better color palette than the one obtained by clustering colors in the RGB space.

The artistic look of the clustered image can be modified by changing the color layers individually. In Figure 4 we present the result obtained by altering the hue, saturation and value of various color layers. The result is a colorful image resembling Shrek.

(a) Shrekified queen (b) Actual Shrek

Figure 4: Shrek.

2 Color Quantization and lookup tables (LUTs) [3 points]

In this exercise we are required to compute Lookup Tables (LUT) for mapping the colors of an image to a reduced color palette. A simplified version of the function shown in the previous section [Figure 2] that does just that is shown below in Figure 5. Notice how variable `cluster_idx` is the Lookup Table. It contains for each pixel the cluster index to which it belongs to. Variable `cluster_color` instead contains the color of each cluster, useful for finally mapping each pixel to its corresponding color in the reduced palette. The resulting image is computed by substituting each index in the 2D LUT matrix with the corresponding 3-dimensional cluster color. Apart from

```
function [img_clustered, cluster_idx, cluster_color ] = clusterColors(img, n_clusters)
    % 1. Cluster
    A = reshape(img, [], 3);
    [cluster_idx, cluster_color] = kmeans(A, n_clusters);

    % Compute clustered image
    img_clustered = reshape(cluster_color(cluster_idx, :), size(img, 1), size(img, 2), 3);
end
```

Figure 5: Matlab function computing the LUT of an image

performing color quantization in the HSV space, we can also perform it in the CIELab space. Results are shown in Figure 6. We clearly notice how the CIELab color quantization shown in Figure 6c yields a better result than the HSV one in Figure 6b. The clustering quality difference is particularly noticeable in the facial region, as the palette colors better resemble the ones founds in the original image. For computing the grayscale LUT, we simply cluster the image in the CIELab space, and then isolate the grayscale information by taking the first channel L* (luminance). The resulting image is shown in Figure 6d.

3 Gaussian and Laplacian Pyramids [4 points]

The Gaussian pyramid of an image is a sequence of images obtained by repeatedly applying Gaussian blurring and downsampling the original image. The Laplacian pyramid is a sequence of images obtained by subtracting from each image in the Gaussian pyramid the upsampled version of the next image in the Gaussian pyramid. Therefore

(a) Original image (b) HSV color space (c) CIELab color space (d) Grayscale-mapped

Figure 6: Quantized images with 32 clusters (colors) in different color spaces

```
%  1.2 Find HSV grayscale quantized image and LUT with 32 colors
cluster_color_gray = cluster_color_lab(:, 1); % Taking Luminance into account
img_clustered_gray = reshape(cluster_color_gray(cluster_idx, :), size(img, 1), size(img, 2), 1);
```

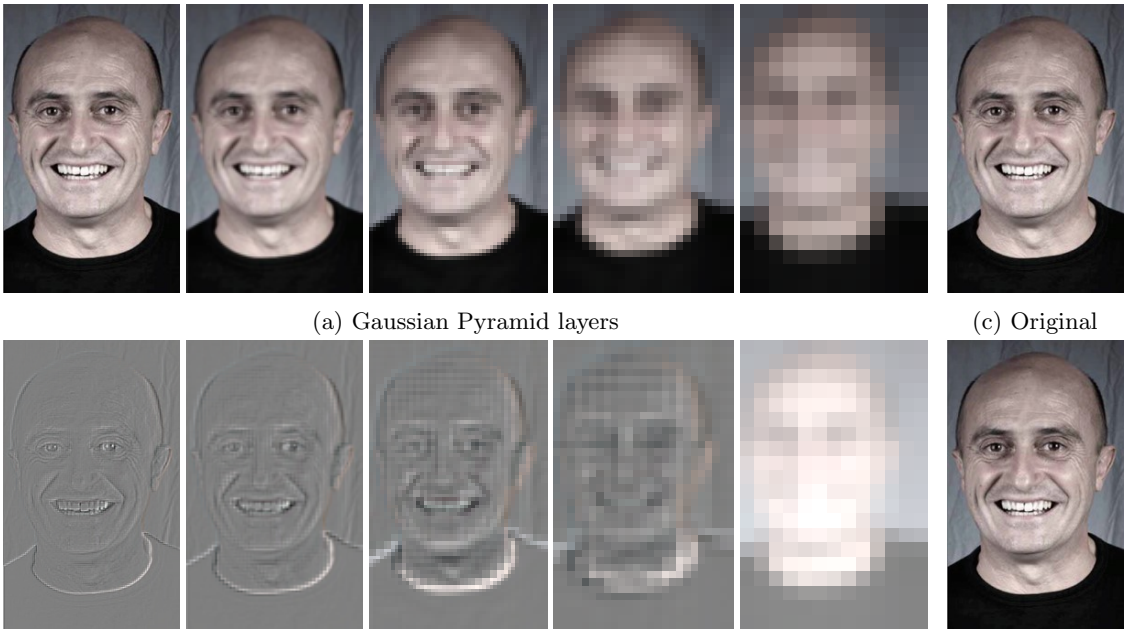
Figure 7: Matlab code for computing the grayscale LUT of an image

each layer of the Lapacian pyramid contains just the details of the corresponding layer in the Gaussian pyramid. To reconstruct the original image we simply add each Laplacian layer to its next upsampled pyramid layer.

The Matlab code used to compute the Gaussian and Laplacian pyramids and reconstruct the original image are shown in Figure 8. The resulting Gaussian and Laplacian pyramids of the image `happy.jpg` are shown in Figure 9. The reconstructed image is shown in Figure 9d.

```
function [gaussian_pyramid, laplacian_pyramid] = compute_pyramids(img, layers)
    layers = layers+1;
    gaussian_pyramid = cell(1,layers);
    gaussian_pyramid{1} = img;

    % compute gaussian pyramid
    for i = 2:layers
        prev_img = gaussian_pyramid{i-1};
        new_img = imgaussfilt(prev_img,1);
        new_img = imresize(new_img,0.5,'nearest');
        gaussian_pyramid{i} = new_img;
    end

    % compute laplatian pyramid
    laplacian_pyramid = cell(1,layers);
    for i = 1:layers-1
        prev_img = gaussian_pyramid{i};
        next_img = gaussian_pyramid{i+1};
        laplacian_pyramid{i} = prev_img - imresize(next_img,[size(prev_img,1),size(prev_img,2)],'nearest');
    end
    laplacian_pyramid{layers} = gaussian_pyramid{layers};

end


function [img] = reconstruct_from_laplacian(laplacian_pyramid)
    layers = size(laplacian_pyramid,2);
    img = laplacian_pyramid{layers};
    for i = layers-1:-1:1
        img = imresize(img,[size(laplacian_pyramid{i},1),size(laplacian_pyramid{i},2)],'nearest') ...
                + laplacian_pyramid{i};
    end
end
```

Figure 8: Matlab functions computing the Gaussian and Laplacian pyramids and reconstructing the original image.



(a) Gaussian Pyramid layers (c) Original

(b) Laplacian Pyramid layers (brightness artifically increased) (d) Reconstructed

Figure 9: Image pyramid decomposition of `happy.jpg`

4 Hybrid Images [3 points]

A fun application of Gaussian and Laplacian pyramids is the creation of hybrid images. In essence, the hu- man visual system is sensitive to high-frequency content of the image at close distances, but the further we move from the image our sensitivity and ability to resolve high-frequency content diminishes, and we better perceive low-spatial frequencies of the image. This can be exploited to create images which combine the high-frequency component of one image with the low-frequency component of another, resulting in an image that changes its appearance depending on the distance from which it is observed.

Figure 10: Hybrid images with different σ values. Lower-Higher σ from left to right.

In Figure 10 we present a set of hybrid images created from `sad.jpg` and `happy.jpg`. These images have been created by combining the low-frequency component of the first image with the high-frequency component of the second image using low-pass and high-pass filters. Different cut-off frequencies have been used from $\sigma = 0.5$ (image at the most left) to $\sigma = 5.0$ (image at the most right).

As can be seen, the "sad" component is dominant in the image at the most left while the "happy" smile becomes more and more visible as the cut-off frequency increases. The image at the most right is dominated by the "happy" smile.

Obviously, the effect varies depending on the size of the image on the screen and the viewing distance.

Figure 11: Showcase of frequency band sensitivity of the human visual system related to the image size.

Figure 11 shows the hybrid image created with $\sigma = 2.5$ in different sizes. As the image gets smaller, the "sad" component becomes more and more visible. This is because the high-frequency component of the image is more visible at close distances, while the low-frequency component is more visible at larger distances.

The central image in Figure 11 displayed on this report printed on A4 paper, has a width of about 2.2cm. By inspecting the printed image, the "happy" smile is viewed optimally when looking at the image from a distance smaller than about 1.21m. By moving away from the image, the "sad" component becomes more and more visible and it is viewed optimally when looking at the image from a distance of about 1.86m or more.

5 Theory: Hybrid Images [3 points]

Consider the task of creating a hybrid image from two input images by combining two levels of Laplacian pyramids. Assume that both images have resolutions 1024×1024 pixels and the size of the hybrid image shown on the screen will be 30×30 centimeters. Additionally, we want the content of one image to be best visible at a distance of 1 meter and the content of the other image to be best visible at a distance of 8 meters.

From a distance of 1m with 1 visual degree of angle, the width of the visual field is 1.75 cm. Since the image is 1024 pixels wide and displayed on the screen with a width of 30 cm, the visual field is $\frac{1.75 \times 1024}{30} \approx 60$ pixels wide. The human eye is more sensitive to some frequencies (around 3 cycles/visual degree) so we need to choose the Laplacian levels accordingly.

Each level of the Laplacian pyramid covers a different frequency band: we can say that "roughly" k^{th} level contains frequencies $\mu \in (0.5^{k+1}, 0.5^k]$ cycles/pixel. So:

$$
\begin{aligned}
0.5^k \cdot 60 &\approx 3 \\
\frac{1}{2^k} \cdot 60 &\approx 3 \\
\frac{60}{2^k} &\approx 3 \\
2^k &\approx 20 \\
k &\approx \log_2(20) \\
k &\approx 4.32
\end{aligned}
$$

The best visible level of the Laplacian pyramid from a distance of 1m is thus the 4^{th} level.

The same reasoning can be applied to the second image, which has to be best visible from a distance of 8m. From a distance of 8m with 1 visual degree of angle, the width of the visual field is 13.96 cm. Since the image is 1024 pixels wide and displayed on the screen with a width of 30 cm, the visual field is $\frac{13.96 \times 1024}{30} \approx 476.6$ pixels wide. As before:

$$
\begin{aligned}
0.5^k \cdot 476.6 &\approx 3 \\
k &\approx 7.31
\end{aligned}
$$

The best visible level of the Laplacian pyramid from a distance of 8m is thus the 7^{th} level.

In conclusion, the chosen levels of the Laplacian pyramid to create a hybrid image that is best visible from a distance of 1m and 8m are 4 and 7 for the first and second images respectively.

6 Bonus: Create your own Instagram Filter! [10 points]

In this last section, we present our own Instagram filter. The filter is composed of three steps: at first, the colors of the image are clustered into 16 clusters using the k-means algorithm and the colors of the image are replaced with the centroids of the clusters. Then, the edges of the original image are detected using the Canny edge detector and the edges are overlayed on top of the image in black. Finally, the image is slightly saturated to give a more cartoonish look.

(a) Original image (b) Quantized image (c) Edges of quantized image (d) Masked and saturated final image

Figure 12: Filter steps

Figure 12 presents the various steps of the filter: the original image (Figure 12a), the image with the colors clustered(Figure 12b), the edges of the image (Figure 12c) and the final image with the edges overlayed and saturated (Figure 12d).

The MATLAB implementation of the filter is shown in Figure 13.

```
function [img, img_clustered, edges, im_stylized] = filter(img)
    % 1. Cluster the image colors
    clusters = 16;
    [img_clustered, cluster_idx, cluster_center ] = clusterColors(img, clusters);

    % 2. Find edges using Canny algorithm
    edges = edge(rgb2gray(img), 'canny', 0.1);

    % 4. Use the edges as a mask to stylize the image
    im_stylized = img_clustered.*~edges;

    % 5. Saturate the colors in HSV space
    im_stylized = rgb2hsv(im_stylized);
    im_stylized(:, :, 2) = im_stylized(:, :, 2)*1.5;
    im_stylized = hsv2rgb(im_stylized);

    edges = gray2rgb(edges, img);
    imshow([img, img_clustered, edges, im_stylized], []);

    function [img_clustered, cluster_idx, cluster_center ] = clusterColors(img, n_clusters)
        % Cluster
        A = reshape(img, [], 3);
        [cluster_idx, cluster_center] = kmeans(A, n_clusters);
        % Compute clustered image
        img_clustered = reshape(cluster_center(cluster_idx, :), size(img, 1), size(img, 2), 3);
    end
end
```

Figure 13: Matlab function applying the *Borderlands 2* filter.